



Programación avanzada en Python

Lección 8: Manipulación de datos

ÍNDICE

Manipulación de datos	1
Presentación y objetivos	1
1. Manipulación de datos	2
Operaciones con archivos	2
2. Trabajando con Pandas	9
Dataframe de Pandas	9
Archivos CSV	11
Operaciones con dataframes.....	13
3. Puntos clave	20

Manipulación de datos

PRESENTACIÓN Y OBJETIVOS

En este capítulo veremos como trabajar con grandes conjuntos de datos. Para ello primero veremos como realizar operaciones directamente con archivos. En la segunda parte veremos como trabajar con dataframe un tipo de datos de la librería Pandas, una de las más utilizadas en Python para el análisis de datos.



Objetivos

En esta lección aprenderás a:

- Realizar operaciones con ficheros.
- Instalar la librería Pandas.
- Realizar operaciones con dataframes.

1. MANIPULACIÓN DE DATOS

Ya hemos hablado de que los datos se pueden introducir a través de una variable y desde el teclado a la vez. Así que es importante pensar en el caso de conjuntos de datos enormes. Esto se hace a través de operaciones con archivos, que pueden manejar una gran variedad de datos.

Operaciones con archivos

Los archivos pueden contener datos enormes, que están en cualquier formato especificado por su extensión. Python maneja fácilmente los archivos con algunas funciones incorporadas.

Los datos pueden ser leídos desde archivos y escritos en archivos. Primero se carga o abre el archivo y a continuación se realizan las operaciones de lectura o escritura, finalmente se cierra el archivo.

Abrir un archivo

Python utiliza una función incorporada `open()` para abrir archivos. El parámetro de la función es la ubicación o el nombre del archivo. La función devuelve un objeto archivo que puede utilizarse para realizar posteriores operaciones de lectura o escritura.

Sintaxis:

Objeto_archivo = open ("carpeta/nombre_archivo. extensión")

Si el archivo se abre desde el directorio actual, sólo se da a la función el nombre del archivo. De lo contrario, se debe especificar la ruta completa.

Ejemplo:

Estoy almacenando datos en un archivo de texto como `sample.txt` como:

```
sample.txt X
1 This is my first file
2 to illustrate the file operations
3 in python.
```

Abrimos el fichero:

```
file = open("/content/sample.txt")
```

El archivo se abre como un objeto de archivo, si tratamos de imprimir el objeto entonces muestra:

```
print(file)
```

```
<_io.TextIOWrapper name='/content/sample.txt' mode='r' encoding='UTF-8'>
```

Si no hay tal archivo en el directorio, lo creará como un nuevo archivo.

El objeto se utiliza para realizar otras operaciones. Así que una vez que el archivo está abierto mediante el objeto creado se puede utilizar en el programa por completo.

Hay dos parámetros opcionales más en la función abrir,

- modo - el modo puede ser especificar para abrir un archivo. Puede ser leer, escribir o añadir. Si el archivo se abre en modo lectura sólo se puede leer del archivo. No se puede hacer ninguna otra operación.

Los modos se representan con caracteres simples. Hay diferentes tipos de modos utilizados en el archivo:

r	Opens a file for reading. (default)
w	Opens a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
x	Opens a file for exclusive creation. If the file already exists, the operation fails.
a	Opens a file for appending at the end of the file without truncating it. Creates a new file if it does not exist.
t	Opens in text mode. (default)
b	Opens in binary mode.
+	Opens a file for updating (reading and writing)

Sintaxis:

File_object = abrir ("ubicación", mode = 'r') o

File_object = open ("location", 'r')

- encode - los caracteres del archivo se codifican por ASCII pero depende de las diferentes plataformas. El tipo de codificación para windows es 'cp1525'. El tipo de codificación para linux es 'utf-8'

Sintaxis :

File_object = open ("location", mode = 'r', encoding = 'utf-8')

```
f = open("test.txt", mode='r', encoding='utf-8')
```

Cerrar un archivo

Después de realizar todas las operaciones, se debe cerrar el archivo. Ello liberará todos los recursos que se utilizó a lo largo del programa. Se hace con la función incorporada, close()

Podemos simplemente cerrarlo con el objeto archivo de la siguiente manera:

```
f = open("test.txt", encoding = 'utf-8')
# perform file operations
f.close()
```

Abriendo fichero con 'with'

El archivo puede abrirse mediante la sentencia 'with'. De esta manera no necesita cerrar el archivo explícitamente.

```
with open("test.txt", encoding = 'utf-8') as f:
    # perform file operations
```

Escribir en el archivo

Para escribir en el archivo, este debe ser abierto en los modos

- w - modo de escritura
- a - modo añadir
- x - modo exclusivo

Si se escribe con el modo de escritura, los nuevos datos se sobrescribirán y los antiguos se borrarán.

La función `write ()` se utiliza para escribir cadenas o sentencias en el archivo.
Veamos un ejemplo:

Este es nuestro archivo de ejemplo

sample.txt X

```
1 This is my first file
2 to illustrate the file operations
3 in python.
```

Vamos a escribir en el archivo con el modo `w`:

```
with open("/content/sample.txt",'w') as file:
    file.write("hello\n")
    file.write("sample program\n")
    file.write("to file operations\n")
```

El archivo se convierte en:

sample.txt X

```
1 hello
2 sample program
3 to file operations
4
```

El modo añadir se utiliza para escribir desde el final del archivo:

Para escribir en el archivo resultante con el modo `a`:

```
with open("/content/sample.txt",'a') as file:
    file.write("open the file\n")
    file.write("read/write\n")
    file.write("close the file\n")
```

El archivo se convierte en:

```
sample.txt X
1 hello
2 sample program
3 to file operations
4 open the file
5 read/write
6 close the file
7
```

'\n' se utiliza para que se produzca una nueva línea en el archivo

Leer de un archivo

El archivo debe ser leído con el modo r y podemos leer las cadenas en el archivo con la función incorporada read().

```
file = open("/content/sample.txt", 'r')
print(file.read())
```

```
hello
sample program
to file operations
```

La función read() tiene un parámetro size, que se utiliza para dar el número de caracteres para la lectura

Sintaxis:

File_object.read (size)

```
file = open("/content/sample.txt", 'r')
print(file.read(4))
```

```
hell
```


tell()

La función tell se utiliza para obtener la posición actual del cursor en un archivo después de leer todo el contenido

```
file = open("/content/sample.txt", 'r')  
print(file.read())
```

```
hello  
sample program  
to file operations
```

```
file.tell()
```

```
40
```

seek()

La función seek se utiliza para mover el cursor a un punto deseado. La posición se da como parámetro.

```
file.seek(20)  
print(file.read())
```

```
to file operations
```

Podemos leer un archivo línea por línea utilizando el bucle for

```
file = open("/content/sample.txt", 'r')  
for line in file:  
    print(line, end = '')
```

```
hello  
sample program  
to file operations
```

readline ()

La función readline se utiliza para leer la línea completa.

```
file = open("/content/sample.txt", 'r')  
file.readline()
```

```
'hello\n'
```

```
file.readline()
```

```
'sample program\n'
```

readlines ()

La función lee las líneas restantes como una lista

```
file.readlines()
```

```
['to file operations\n']
```

```
file = open("/content/sample.txt", 'r')  
file.readlines()
```

```
['hello\n', 'sample program\n', 'to file operations\n']
```

2. TRABAJANDO CON PANDAS

Existe un formato de datos utilizado para manipular datos mediante la librería pandas que se llama dataframe.

Dataframe de Pandas

El Dataframe de Pandas es un dato tabular con ejes bidimensionales (filas y columnas) que están etiquetados y pueden o no tener valores.

Pandas se puede instalar en el entorno Python fácilmente usando pip as:

```
pip install pandas
```

Si cuando vallamos a instalar el paquete ya está instalado muestra los requisitos existentes:

```
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (1.1.5)  
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas) (2.8.1)  
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.7/dist-packages (from pandas) (1.19.5)  
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas) (2018.9)  
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
```

Una vez instalado se puede importa el paquete pandas:

```
import pandas as pd
```

Pandas Dataframe puede ser creado a partir de cualquier de estructuras de datos, diccionario, etc. Hay algunos métodos del paquete pandas que se utilizan para ello.

pandas.DataFrame ()

Parámetros:

- data - se pueden dar datos en forma lista, diccionario, lista de diccionario y ndarray
- index - si se especifica el índice, se mostrará la tabla con índice
- column - si se especifica el nombre de la columna, se obtendrá la columna correspondiente.
- dtype - se puede dar un solo dtype. Por defecto es ninguno.
- copy - los datos pueden ser copiados de las entradas. Por defecto será falso.

Una lista se puede convertir en Dataframe utilizando el método pandas:

```
import pandas as pd
list = ['hello', 'python', 'dataframe']

df = pd.DataFrame(list)
print(df)
```

	0
0	hello
1	python
2	dataframe

Esto dará como resultado un Dataframe con 3 filas y una columna.

Convirtiendo un diccionario a Dataframe:

```
import pandas as pd

data = {'Name': ['xyz', 'abc', 'pqr', 'zxc', 'fgh'],
        'Place': ['asdf', 'sddfg', 'dfghj', 'lkjhg', 'edfgt']}

df = pd.DataFrame(data)
df
```

	Name	Place
0	xyz	asdf
1	abc	sddfg
2	pqr	dfghj
3	zxc	lkjhg
4	fgh	edfgt

Una sola columna puede ser seleccionada utilizando el nombre de la columna:

```
df[['Name']]
```

	Name
0	xyz
1	abc
2	pqr
3	zxc
4	fgh

Los datos también se pueden cargar desde grandes conjuntos de datos como archivos csv, excel y desde una base de datos sql.

Archivos CSV

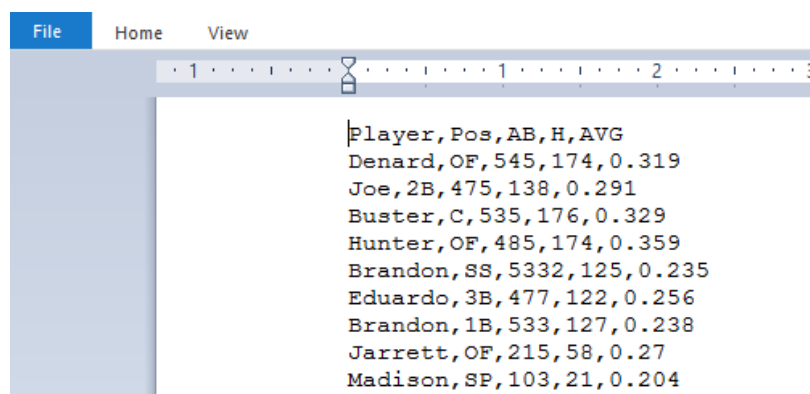
El formato de fichero de datos más común es csv. Es un archivo de valores separados por comas que se almacena como una lista de datos que se separan con comas y cuando se abre en Excel u hoja de cálculo se convierte en un formato tabular.

El archivo csv se puede abrir con el atributo `read_csv`:

```
pandas.read_csv()
```

se pasa como parámetro el nombre del archivo o la ubicación. Pongamos un ejemplo:

Aquí vemos un archivo csv creado con algunos datos, se abrirá en un editor de texto para obtener la vista de valores separados por comas:



The screenshot shows a text editor window with a menu bar (File, Home, View) and a scroll bar. The text content is a CSV file with 10 rows of baseball player statistics. The first row contains column headers: Player, Pos, AB, H, AVG. The subsequent rows contain data for Denard, Joe, Buster, Hunter, Brandon, Eduardo, Brandon, Jarrett, and Madison, each with their position, at-bats, hits, and batting average.

Player	Pos	AB	H	AVG
Denard	OF	545	174	0.319
Joe	2B	475	138	0.291
Buster	C	535	176	0.329
Hunter	OF	485	174	0.359
Brandon	SS	5332	125	0.235
Eduardo	3B	477	122	0.256
Brandon	1B	533	127	0.238
Jarrett	OF	215	58	0.27
Madison	SP	103	21	0.204

Cuando se abre en una hoja de cálculo, estará en un formato tabular:

	A	B	C	D	E	F
1	Player	Pos	AB	H	AVG	
2	Denard	OF	545	174	0.319	
3	Joe	2B	475	138	0.291	
4	Buster	C	535	176	0.329	
5	Hunter	OF	485	174	0.359	
6	Brandon	SS	5332	125	0.235	
7	Eduardo	3B	477	122	0.256	
8	Brandon	1B	533	127	0.238	
9	Jarrett	OF	215	58	0.27	
10	Madison	SP	103	21	0.204	
11						

Este fichero se puede abrir como un Dataframe de pandas con el método `read_csv` para posteriormente poder manipular los datos:

```
import pandas as pd
csv = pd.read_csv('/content/file.csv')
csv
```

	Player	Pos	AB	H	AVG
0	Denard	OF	545	174	0.319
1	Joe	2B	475	138	0.291
2	Buster	C	535	176	0.329
3	Hunter	OF	485	174	0.359
4	Brandon	SS	5332	125	0.235
5	Eduardo	3B	477	122	0.256
6	Brandon	1B	533	127	0.238
7	Jarrett	OF	215	58	0.270
8	Madison	SP	103	21	0.204

Comprobemos el tipo de datos leídos del fichero csv

```
type(csv)

pandas.core.frame.DataFrame
```

Podemos cargar los datos haciendo que cualquier columna en particular sea una columna índice utilizando el parámetro como:

indexcol= 'nombre de la columna'

```
import pandas as pd
csv = pd.read_csv('/content/file.csv', index_col='Pos')
csv
```

	Player	AB	H	AVG
Pos				
OF	Denard	545	174	0.319
2B	Joe	475	138	0.291
C	Buster	535	176	0.329
OF	Hunter	485	174	0.359
SS	Brandon	5332	125	0.235
3B	Eduardo	477	122	0.256
1B	Brandon	533	127	0.238
OF	Jarrett	215	58	0.270
SP	Madison	103	21	0.204

Operaciones con dataframes

Indexación booleana

Si queremos extraer datos del Dataframe basados en algunas condiciones, podemos utilizar la indexación booleana.

Se puede recuperar las filas de un criterio particular por el método 'loc':

Dataframe.loc[(condición [nombre_columna])]

Veamos un ejemplo:

Del Dataframe anterior, queremos tomar la lista de jugadores cuya media es inferior a 0.3, el método loc se puede definir como:

```
csv.loc[csv["AVG"] < 0.3]
```

Y el resultado,

	Player	Pos	AB	H	AVG
1	Joe	2B	475	138	0.291
4	Brandon	SS	5332	125	0.235
5	Eduardo	3B	477	122	0.256
6	Brandon	1B	533	127	0.238
7	Jarrett	OF	215	58	0.270
8	Madison	SP	103	21	0.204

Podemos almacenar este resultado como un nuevo Dataframe.

El método loc puede tomar más de una condición con operadores lógicos.

Por ejemplo, si queremos extraer los jugadores con media inferior a 0.3 y la altura debe ser superior a 100, entonces se podría hacer de la siguiente manera:

```
csv.loc[(csv["AVG"] < 0.3) & (csv["H"]>100)]
```

	Player	Pos	AB	H	AVG
1	Joe	2B	475	138	0.291
4	Brandon	SS	5332	125	0.235
5	Eduardo	3B	477	122	0.256
6	Brandon	1B	533	127	0.238

Aplicando funciones

Se puede aplicar una función al Dataframe de pandas. La función puede ser incorporada o definida por el usuario. La función puede aplicarse al marco con el método `apply()`, sus parámetros son:

- Función - la función que se aplica al conjunto de datos.
- Eje - si el eje es 0, la función se aplica a través de las filas y si el eje es 1, se aplica a las columnas

Veamos un ejemplo:

```
data = {'A': [3,2,4,1,3],  
        'B': [33,55,22,44,11]}  
  
df = pd.DataFrame(data)  
df
```

	A	B
0	3	33
1	2	55
2	4	22
3	1	44
4	3	11

Si queremos aplicar la función incorporada `sum`:

```
df.apply(sum, axis = 0)  
  
A      13  
B     165  
dtype: int64
```

Aquí se utiliza la función incorporada `sum`. En lugar de eso se puede utilizar cualquier función incorporada.

Por ejemplo, si se quiere obtener el mínimo de dos columnas se define una función para realizar la función min:

```
def minimum(x):
    return min(x)

df.apply(minimum, axis = 0)

A      1
B     11
dtype: int64
```

Crosstab ()

Es una función que se utiliza para ver los datos desde otra perspectiva. Ayuda a validar las columnas.

```
pd.crosstab(df.A,df.B)
```

	B	11	22	33	44	55
A						
1	0	0	0	1	0	
2	0	0	0	0	1	
3	1	0	1	0	0	
4	0	1	0	0	0	

Veamos un ejemplo con el Dataframe csv. La comparación de sus dos columnas, H y AVG será:

```
pd.crosstab(csv.H,csv.AVG)
```

	AVG	0.204	0.235	0.238	0.256	0.270	0.291	0.319	0.329	0.359
H										
21	1	0	0	0	0	0	0	0	0	0
58	0	0	0	0	1	0	0	0	0	0
122	0	0	0	1	0	0	0	0	0	0
125	0	1	0	0	0	0	0	0	0	0
127	0	0	1	0	0	0	0	0	0	0
138	0	0	0	0	0	1	0	0	0	0
174	0	0	0	0	0	0	1	0	1	
176	0	0	0	0	0	0	0	1	0	

Combinando dataframes

Las columnas de los dataframes se pueden fusionar utilizando la función `merge()`. Los parámetros son los dataframes que se van a fusionar y hay un parámetro más 'on', que sirve para especificar la columna común en el Dataframe.

Ejemplo: hay dos marcos de datos con una columna común, nombre

```
data = {'Name': ['xyz', 'abc', 'pqr', 'zxc', 'fgh'],
        'Place': ['asdf', 'erty', 'dfghj', 'tyuio', 'edfgt']}

df1 = pd.DataFrame(data)

data1 = {'Name': ['xyz', 'abc', 'pqr', 'zxc', 'fgh'],
        'Age': ['23', '34', '18', '55', '43']}

df2 = pd.DataFrame(data1)
```

df1			df2		
	Name	Place		Name	Age
0	xyz	asdf	0	xyz	23
1	abc	erty	1	abc	34
2	pqr	dfghj	2	pqr	18
3	zxc	tyuio	3	zxc	55
4	fgh	edfgt	4	fgh	43

La columna edad y lugar quieren ser fusionados y almacenados en un nuevo marco de datos, podemos utilizar la función de `merge()` con la columna común, nombre:

```
df3 = pd.merge(df1, df2, on="Name")

df3
```

	Name	Place	Age
0	xyz	asdf	23
1	abc	erty	34
2	pqr	dfghj	18
3	zxc	tyuio	55
4	fgh	edfgt	43

Ordenando dataframes

Un dataframe puede ser ordenado con dos funciones:

- `sort_values()` - ordena los dataframes por una o más columnas.
- `sort_index()` - ordena los dataframes por uno o más índices de fila.

Ejemplo:

Desde el dataframe `df3` anterior, se puede ordenar con la columna `edad` utilizando el método `sort_values()`:

```
df3.sort_values("Age")
```

	Name	Place	Age
2	pqr	dfghj	18
0	xyz	asdf	23
1	abc	erty	34
4	fgh	edfgt	43
3	zxc	tyuio	55

El dataframe `csv` con el índice `Pos` puede ser ordenado con la función `sort_index()`

```
csv = pd.read_csv('/content/file.csv', index_col='Pos')
csv
```

	Player	AB	H	AVG
Pos				
OF	Denard	545	174	0.319
2B	Joe	475	138	0.291
C	Buster	535	176	0.329
OF	Hunter	485	174	0.359
SS	Brandon	5332	125	0.235
3B	Eduardo	477	122	0.256
1B	Brandon	533	127	0.238
OF	Jarrett	215	58	0.270
SP	Madison	103	21	0.204

```
csv.sort_index()
```

	Player	AB	H	AVG
Pos				
1B	Brandon	533	127	0.238
2B	Joe	475	138	0.291
3B	Eduardo	477	122	0.256
C	Buster	535	176	0.329
OF	Denard	545	174	0.319
OF	Hunter	485	174	0.359
OF	Jarrett	215	58	0.270
SP	Madison	103	21	0.204
SS	Brandon	5332	125	0.235

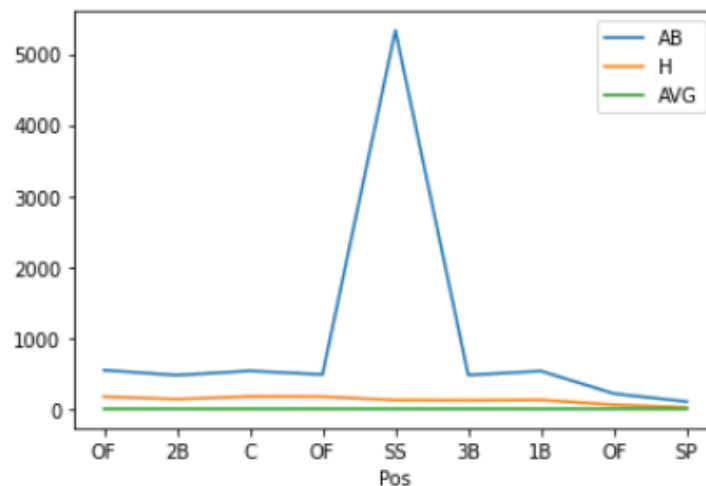
Plotting

Los dataframes se pueden dibujar utilizando el método `plot()`. Cuando llamamos a este método se traza con el índice como eje x y cada columna se traza con diferentes líneas de diferentes colores.

El dataframe indexado anterior `csv`, puede ser trazado de la siguiente manera:

```
csv.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcdd758b990>
```



El índice Pos se toma como eje X y las demás columnas, AB, H, AVG se trazan con diferentes líneas.

3. PUNTOS CLAVE

- | Python puede leer directamente desde **ficheros**. Esto es muy útil cuando trabajamos con grandes **conjuntos de datos**.
- | **Pandas** es una de las librerías más utilizadas en **Python** para el manejo de grandes **conjuntos de datos**.
- | Existe un formato de datos utilizado para manipular datos mediante la librería pandas que se llama **dataframe**.
- | El **Dataframe** de Pandas es un dato tabular con ejes bidimensionales (filas y columnas) que están etiquetados y pueden o no tener valores.

